Apple Assembly Line

$1.50

Volume 2 -- Issue 12 September, 1982

In This Issue...

Current Advertising Rates

Sorry, it is going up again. For the October 1982 issue the price will be $75 for a full page, $40 for a half page. To be included, I must receive your camera-ready copy by September 20th.

What if you move?

We mail the Apple Assembly Line by bulk mail, unless you have paid for First Class or Overseas postage. If you move, the post office will NOT forward AAL to your new address. Please let us know your new address as soon as you find out what it will be, so you will not miss a single issue!

Quarterly Disks

As you no doubt know, every three months we gather all the source code printed during the quarter on one disk. You can save countless hours of typing and proofreading for only $15 per quarter. Some have elected to establish a standing order with their credit card, or even to prepay for a year at a time.

New Products

ES-CAPE: For really painless Applesoft programming, you need a complete line editor, global search and replace, automatic line numbers, and keyboard macros. At least. ES-CAPE gives you all these and more!

The retail price is $60, but AAL subscribers can get it for only $40 until the end of September. Hurry!

We wrote a nice little reference manual of about 22 pages, but ES-CAPE is so easy to use and remember that you won't need the book very long!

If you already purchased AED II (the earlier version of this editor), Bill Linn has an upgrade offer: Send him your disk plus $10, and you will get the new versions (both regular and language card), the manual, and the reference card.

68000 Macro Cross Assembler: Not content with producing only three cross assemblers based on the S-C Macro Assembler, Bobby Deen has now completed the biggest one of all! This one costs $50, and allows you to assemble Motorola 68000 source programs in your Apple, with all the friendly features of the S-C package.

SYNASSEMBLER: Synapse Software has just started marketing a conversion of the S-C Assembler II Version 4.0 for the Atari 800 or 400. You need 48K RAM and at least one disk drive. The conversion was done by Steve Hales, of Livermore, California. He added global replace and copy commands, so this version falls somewhere between the Apple version 4.0 and the new Macro version. It assembles at about 6500 lines per minute, which is from 50 to over 100 times faster than the Atari ASM/ED program.

Since the Atari does not have nice monitor commands built-in, like the Apple does, Steve added a complete set of monitor commands to SYNASSEMBLER. They look exactly like the Apple monitor commands, except that he added some new ones to allow reading and writing a range of disk sectors, delete the tape I/O commands, and included the old Step and Trace commands which were in Apples before the Autostart ROM.

The price is only $49.95 on disk. A ROM version is available by special order for $89.95. I will carry these, if you want to order from me.

Assembler Directives.....Bob Sander-Cederlof

Of all the Apple assemblers on the market, it seems that no two have exactly the same list of assembler directives. Directives, also called "pseudo-ops", are used to control the assembly process and to define data in your programs. When you see a listing of an assembly language program in a magazine, or in this newsletter, or in a book on 6502 programming, you may have to translate the directives to fit the assembler you own.

All directives in the S-C Macro Assembler begin with a period. This helps to distinguish them visually from 6502 and SWEET-16 opcodes. This same convention is used by Carl Moser's (Eastern House Software) MAE assembler, by the MOS Technology and Rockwell assemblers, and some others. Most other assemblers use 3- or 4-character mnemonics beginning with a letter. Which combination of letters cause the assembler to perform a particular function is not standardized at all, but there are enough similarities to make programs readable once you learn the general techniques.

What follows is an alphabetical listing of all the directives I have encountered in various manuals and magazine-published programs. The assemblers represented are coded like this:

B = Big Mac	SC= S-C Macro Assembler
K = DOS Tool Kit	T = TED II
L = Lisa	W = Weller's Assembler
M = Merlin	

In each case I have given a brief description of the directive, and tried to show how to do the same thing in the S-C Macro Assembler. I suggest looking up the S-C directives in the reference manual if you are not sure exactly how to use them.

ADR ADdRess L
Stores the expression as an address, low-order byte first.
SC: Use .DA directive

ASC ASCii string definition L K T B M
SC: Use .AS or .AT directives.

AST ASTerisks T B M
Prints the number of asterisks specified on the listing. Used to save space in the source file.
SC: Not needed, because SC compresses repeated characters automatically.

BLK BLinKing characters L
Generates a string of characters in Apple's FLASH code.
SC: Not available, but a combination of .AS and .HS directives will do the job.

BYT BYTe data L
Define data value, storing low-order byte only.
SC: Use .DA with "#" before value.

CHK CHecKsum B M
SC: Not available

CHN CHaiN to next source K
SC: Use .IN directive.

CHR Set CHaR for REP directive K
Used to create fancy comments with repeated strings; saves space in source file.
SC: Not necessary, because SC compresses repeated characters automatically.

.DA DAta definition L SC
Apparently Randy borrowed this one from me. (See the reviews he wrote in Call APPLE some time ago.)

DA Define Address T B M
Defines a 16-bit value with low-byte first.
SC: Use .DA directive.

DATA DATA definition W
Defines numeric and ASCII data bytes
SC: Use .DA directive, preceding each value with "#".

DB Data Byte T
Defines a data value, only using the low byte of the expression.
SC: Use .DA directive with "#" before the expression.

DBL DouBLe precision data W
Defines 16-bit data values.
SC: Use .DA directive.

DBY Double BYte data L
Generates a 16-bit value and stores it high-byte first.
SC: Not directly available, but use .DA as follows:
```
.DA /expression,#expression
```

DCI Define Characters Immed L K T B M
Stores string with sign bit of last byte opposite that of the rest of the bytes.
SC: Use .AT directive.

DCM DOS CoMmand L
Issue a DOS command during assembly. Usually used to BSAVE a section of the generated object code.
SC: Use .TF directive

DDB Define Double Byte K B M
Defines a 16-bit value which is stored with the high-byte first.
SC: Not directly available, but use .DA as follows:
```
.DA /expression,#expression
```

DEND Dummy END K
Marks end of a dummy section (see DSECT).
SC: Not available

```
DFB     DeFine Byte             K B M
Defines one or more bytes.
SC:  Use .DA, preceding each expression with "#".

DFC     DeFine Character        L (old version)
Data definition, byte expression list
SC:  Use .DA directive, preceding each expression with "#".

DFS     DeFine Storage          L
Reserve a block of bytes.  An optional second operand will
cause the reserved bytes to be set to the specified value.
SC:  Use .BS directive.  No option to set the reserved bytes to
a specified value.

DO      DO                      K B M
Start a conditional assembly block.
SC:  Use .DO directive.

DPH     DePHase                 L
Terminates a PHS directive.
SC:  Not available.

DS      Data Storage            T K B M
Reserve a block of bytes.
SC:  Use .BS directive.

DSC     Data SeCtion            L (old version)
Not sure what this was for.

DSECT   Dummy SECTion           K
Starts a block in which the object code bytes are not written
on the output file.
SC:  Not available.

DW      Define Word             K T
Defines a 16-bit value, with the low-byte stored first.
SC:  Use .DA directive.

.EL     ELse                    L SC
For conditional assembly.

ELSE    ELSE                    K B M
For conditional assembly, toggles the truth value from the DO
directive.
SC:  Use .ELSE directive.

END     END of program          L T B M W
Most assemblers REQUIRE an "END" directive at the end of the
source code.  S-C allows it but does not require it.
SC:  Use .EN directive.

ENTRY   ENTRY                   K
Indicates a symbol is to be made reference-able from other
separately-assembled modules.  To be used by a linking loader
program, which Apple does not provide.
SC:  Not available.
```

QUICKTRACE

relocatable program traces and displays the actual machine operations, *while* it is running without interfering with those operations. Look at these *FEATURES:*

***Single-Step** mode displays the last instruction, next instruction, registers, flags, stack contents, and six user-definable memory locations.*

***Trace** mode gives a running display of the Single-Step information and can be made to stop upon encountering any of nine user-definable conditions.*

***Background** mode permits tracing with no display until it is desired. Debugged routines run at near normal speed until one of the stopping conditions is met, which causes the program to return to Single-Step.*

***QUICKTRACE** allows changes to the stack, registers, stopping conditions, addresses to be displayed, and output destinations for all this information. All this can be done in Single-Step mode while running.*

***Two optional display formats** can show a sequence of operations at once. Usually, the information is given in four lines at the bottom of the screen.*

***QUICKTRACE** is completely transparent to the program being traced. It will not interfere with the stack, program, or I/O.*

***QUICKTRACE** is relocatable to any free part of memory. Its output can be sent to any slot or to the screen.*

***QUICKTRACE** is completely compatible with programs using Applesoft and Integer BASICs, graphics, and DOS. (Time dependent DOS operations can be bypassed.) It will display the graphics on the screen while **QUICKTRACE** is alive.*

***QUICKTRACE** is a beautiful way to show the incredibly complex sequence of operations that a computer goes through in executing a program*

See these programs at participating Computerland and other fine computer stores.

QUICKTRACE *$50*

Is a trademark of Anthro-Digital, Inc.

Written by John Rogers

Anthro - Digital Software, Inc.
P.O. Box 1385 Pittsfield, MA 01202

```
EOM     End Of Macro                B M
Marks end of a macro definition.
SC:  Use .EM directive.

EPZ     Equate Page Zero            L
label   EPZ expression
Defines the label to have the value of the expression, which
must be from $00 to $FF.  When EPZ-defined labels are used in
address fields, zero-page addressing mode will be used whenever
possible.
SC:  Use .EQ directive.  SC automatically uses page-zero mode
whenever possible.

EQU     EQUate                      L T K B M W
label   EQU expression
Defines the label to have the value of the expression during
the assembly process.
SC:  Use .EQ directive.

ESP     End ScratchPad              W
Works with SPD to bracket a data section.
SC:  Not needed.

EXP     EXPansion of macros         B M
Controls whether macro expansion code is printed or not on the
output listing.
SC:  Use .LIST directive.

EXTRN   EXTeRNal                    K
Indicates that a label is externally defined.  To be used with
a linking loader program, which Apple does not provide.
SC:  Not available.

.FI     end of conditional          L SC

FIN     end of conditional          K B M
SC:  Use .FIN directive.

FLS     FLaSH                       B M
Define a string in flashing mode.
SC:  Not available, but a combination of .AS and .HS directives
will do the job.

GEN     GENerate code listing       L
Turns on listing of all object code bytes.
SC:  Not available, object code listing is always on.

HBY     High BYte                   L
Define one-byte data value, storing only the high-byte of an
expressions value.
SC:  Use .DA directive, writing "/" before the value.

HEX     HEXadecimal data            L T B M
label   HEX hexstring
SC:  Use .HS directive.
```

ICL InCLude L
Really is a CHAIN to next source file.
SC: Use .IN directive.

.IF conditional assembly L
SC: Use .DO directive.

INV INVerted characters L B M
Generates a string of characters in Apple's INVERSE screen code.
SC: Not available, but you can convert to hexadecimal and use .HS directive.

LET label reassignment L
Same as EQU, except label can be redefined during assembly.
SC: Not available.

LST LiST option L T K B M
Turn assembly listing on or off.
SC: Use .LIST directive.

MAC MACro definition B M
Start a macro definition.
SC: Use .MA directive.

MSB Most Signficant Bit K
Controls whether the ASC directive generates bytes with the first bit set or clear.
SC: Use .AS or .AT directives with or without the "-" before the first delimiter to indicate the MSB value.

NLS No List option L
Turn assembly listing off.
SC: Use .LIST OFF directive.

NOG NO Generate L
Turns off listing of all but first three bytes of any particular source line.
SC: Not available.

OBJ OBJect address L T B M
Set actual memory address for assembled object code to be stored in.
SC: Use .TA directive.

ORG ORiGin L T K B M
Set memory address program will execute at.
SC: Use .OR directive.

PAG PAGe eject on listing L T B M
Sends control-L to listing device.
SC: Use .PG directive.

PAGE PAGE eject on listing K
Sends control-L to listing device.
SC: Use .PG directive.

```
PAU    PAUse and force error   L B M
SC:  Not available.

PHS    PHaSe                   L
Allows setting ORG without changing OBJ.  Terminated with DPH.
SC:  Not available.

PMC    Present MaCro           B M
Opcode to call a macro.
SC:  Not needed, macros are called by their own names.

PR#    Select printer slot     T
SC:  Select before assembly begins using DOS "PR#slot" command,
or SC "PRT" command.

REL    RELocatable object      K
Causes assembler to generate a relocation dictionary at the end
of the object file, for use by Apple's relocating loader.
SC:  Not available.

REM    REMark                  W
Used to indicate a comment line.
SC:  Use "*" in first column of label field.

REP    REPeated character      K
Generates a string of repetitions of the current CHR value on
the output listing.  Used to save space in the source file.
SC:  Not needed, because SC automatically compresses repeated
characters.

SAV    SAVe object code        B M
SC:  Use .TF directive.

SBTL   SuBTitLe                K
Provides a title line for the top of each page of the output
listing.
SC:  Use .TI directive.

SKP    SKiP lines              K B M
Leaves a specified number of blank lines in the output listing.
SC:  Not available.

SPD    ScratchPaD              W
Works with ESP to bracket a data section.
SC:  Not needed.

STR    STRing                  L
Similar to Lisa's ASC except the first byte output is the
length of the string.
SC:  labela .DA #labelb
           .AS /string/
     labelb .EQ *-labela-1

SYM    SYMbols                 T
Produces a symbol cross-reference table at end of assembly.
SC:  Not available, but can use Rak-Ware's XREF utility
program.
```

```
TITL   TITLe
TTL    TiTLe                    L
```
Generates title line at top of each page of listing.
SC: Use .TI directive.

```
TR     TRuncate object listing  B M
```
Limit listing of object code to 3 bytes per source line.
SC: Not available.

```
USR    USeR directive           L
```
An extra entry in the directive table for the user to use as he sees fit.
SC: Use .US directive.

```
;      comment indicator        L
```
SC: If ";" was in first column, use "*" instead. If in later column, no special character is needed.

```
=      equate                   B M others
```
If written with label on left, this is the same as EQU and .EQ directives. If written with "*" on the left, it is the same as ORG and .OR directives.

```
<<<                             B M
```
Alternate syntax for EOM.
SC: Use .EM directive.

```
>>>                             B M
```
Alternate syntax for PMC.
SC: Not needed, because macros are called by their own names.

Directives in Roger Wagner's Book

If you have been trying to learn using the S-C Assembler with Roger's book "Assembly Lines: The Book", you may have been frustrated by his use of several assembler directives. He discusses directives on pages 16-18, and 55.

On page 16, the first example of the use of directives has two errors. Lines 6 and 7 are written:

```
     6  OBJ EQU $300
     7  ORG EQU $300
```

```
But they should be:  6    OBJ $300
                     7    ORG $300
```

That is, OBJ and ORG are directives, not labels. The top two lines on page 21 are also incorrect, in that the ORG and OBJ directives were typeset to look like labels; they should be moved over to the opcode column, and the "$300" values to the operand column.

In all, Roger uses only five directives in his book: OBJ, ORG, EQU, ASC, and HEX. To use his programs in the S-C assembler, change:

```
 From                    To
----------------        ----------------
label EQU value         label .EQ value
label HEX hexdigits     label .HS hexdigits
      HEX hexdigits           .HS hexdigits
label ASC "characters" label .AS -"characters"
      ASC "characters"        .AS -"characters"
      OBJ $300 or $302     omit this line
      ORG $300 or $302        .OR $300 or $302
```

Note that the normal translation of "OBJ" is ".TA"; however, when the address is the same as the ORG/.OR address, it is not necessary to use OBJ/.TA. Furthermore, in the S-C Assemblers you must put the ".OR" line BEFORE the ".TA" line. In Roger's examples these two lines are reversed.

Relocatable Ampersand-Vector.....Steve Mann

In recent issues of AAL there have been a variety of routines to produce relocatable code. The BSR, BRA and LEAX opcodes in the June issue and the run-anywhere subroutine calls in the July issue are two examples.

However, in making some of my code relocatable, I encountered a new problem with routines that interface with Applesoft programs through the & command. The problem is that the routine doesn't know what address to place in the & jump vector because that address may change with each run.

A rather inelegant solution is to derive the address from Applesoft's pointers, then POKE it into the & vector before calling it. What I wanted was a method to determine the correct address from within the code itself, in much the same way that a non-relocatable program sets up the vector:

```
1000         LDA #$4C
1010         STA AMPER.VECTOR
1020         LDA #START
1030         STA AMPER.VECTOR+1
1040         LDA /START
1050         STA AMPER.VECTOR+2
1060 *
1070 START
```

I have written a short routine which will handle the initialization at the beginning of relocatable programs, as long as the program's entry point immediately follows, as in the sample program listed below.

The routine works by first jumping to the subroutine at $FF58, which is simply an RTS instruction. As Bob explained in the July AAL, this places the return address on the stack and then pops it back off again. The return address can then be found by reading the first two open bytes below the stack. The TSX instruction in line 1100 loads the offset to those two bytes into the X-register. Lines 1110-1130 load the bytes into the A- and Y-registers.

Now we have the address of the third byte of the JSR RETURN instruction - the MSB in Y and the LSB in A. What we need is the address of the program's entry point, which corresponds to the label START. To get that address, we must add in the length of the rest of the SETUP routine, that is, the difference between the address at START and the address in the Y- and A-registers.

This is handled in lines 1140-1170. Line 1150 adds the offset ($1B for this particular routine) to the low byte of the base address. The extra 1 in the ADC intruction is necessary because the address in Y and A is one less than the actual return address (corresponding to .1). Lines 1160-1170 check for a carry and adjust the high byte if necessary. The entry point address is then saved in the ampersand vector at $3F5-$3F7.

AMPERWARE

THE ULTIMATE ENHANCEMENT TO APPLESOFT* BASIC

There are a number of Applesoft enhancement packages on the market which use the ampersand reserved word to gain access to assembly language routines from BASIC. None can match the ease of use, speed or powerful features of **Amperware.** Just BLOAD AMPERWARE and you have access to an extended instruction set which will make your programs smaller, faster, more professional and easier to write. No knowledge of assembly language is required and there are not POKEs, PEEKs, CALLs or limitations on variables.

—— FEATURES ——

FLEXIBLE DATA ENTRY -- The &INPUT command allows rapid creation of video forms with line editing (insert/delete character, clear to end, etc.), upper/lower case and numeric error checking. Control codes handling is easy with the &ON CTRL GOTO statement.

RAPID DISK I/O -- Read or write numeric or string information to the disk is binary up to 20 times faster than in BASIC. Compact storage format saves 25-75% of disk space. &READ/&WRITE whole arrays or even lists of arrays with one statement.

FORMATTED OUTPUT -- &PRINT WITH will print in regular, exponential or accounting format (negative in parentheses). Complete control of: significant digits; decimal sign; dollar sign; comma placement; fill characters (blank, zero or asterisk). Handles text in the format and will place text into format fields with right/left justification or centering.

MANY OTHERS -- Sort 8-10 times faster (direct or indexed); delete arrays; rapid substring search; GOTO/GOSUB to computed line number; simple text window control; hardware independent cursor and screen control; etc.

ALL THIS FOR ONLY $49.95
add $1.50 shipping and handling

SCIENTIFIC SOFTWARE PRODUCTS, INC.
3171 Donald Avenue
Indianapolis, IN 46224

IN residents add 4% tax VISA and MC accepted

* Applesoft is a trademark of Apple Computer Co., Inc.

```
                    1000 *--------------------------------
0100-               1010 STACK        .EQ $100
03F5-               1020 AMPER.VECTOR .EQ $3F5
FF58-               1030 RETURN       .EQ $FF58
FC58-               1040 HOME   .EQ $FC58
                    1050 *--------------------------------
                    1060         .OR $300
                    1070         .TF B.AMPEXAMPLE
                    1080 *--------------------------------
0300- 20 58 FF      1090 SETUP   JSR RETURN          PUT CURRENT ADDR ON STACK
0303- BA            1100 .1      TSX                 GET STACK POINTER FOR OFFSET
0304- BC 00 01      1110         LDY STACK,X         MSB OF ADDR ON STACK
0307- CA            1120         DEX
0308- BD 00 01      1130         LDA STACK,X         LSB
030B- 18            1140         CLC
030C- 69 1B         1150         ADC #START-.1+1     OFFSET TO ENTRY POINT
030E- 90 01         1160         BCC .2
0310- C8            1170         INY                 (Y) IS HI BYTE
0311- 8D F6 03      1180 .2      STA AMPER.VECTOR+1  LSB OF ENTRY ADDRESS
0314- 8C F7 03      1190         STY AMPER.VECTOR+2  MSB
0317- A9 4C         1200         LDA #$4C            JMP OPCODE
0319- 8D F5 03      1210         STA AMPER.VECTOR
031C- 60            1220         RTS
                    1230 *--------------------------------
031D- 20 58 FC      1240 START   JSR HOME      CLEAR SCREEN
0320- EA            1250         NOP           DO WHATEVER
0321- EA            1260         NOP           YOU LIKE
0322- 60            1270         RTS
```

The same principle can be used to set up the monitor's control-Y vector at $3F8-$3FA. As a matter of fact, I usually use a macro with conditional assembly to set up whichever vector I need. Here's the macro:

```
1000          .MA VECTOR
1010          JSR $FF58
1020 :1       TSX
1030          LDY $100,X
1040          DEX
1050          LDA $100,X
1060          CLC
1070          ADC #:3-:1+1
1080          BCC :2
1090          INY
1100 :2       .DO ']1='Y    CTRL-Y?
1110          STA $3F9
1120          STY $3FA
1130          LDA #$4C
1140          STA $3F8
1150          .ELSE         OR &?
1160          STA $3F6
1170          STY $3F7
1180          LDA #$4C
1190          STA $3F5
1200          .FIN
1210          RTS
1220 :3
1230          .EM
```

Just include this definition at the beginning of your program. Then macro can then be called like this:

```
2000         >VECTOR,Y
2010 START   ...
```

to set the control-Y vector, or like this:

```
2000         >VECTOR,&
2010 START   ...
```

to set the ampersand vector. (Actually any character other than Y will result in setting the & vector.)

(Note: When I showed this macro to Bob I asked him if the .DO in line 1100 would really work. He looked at it for a minute and said, "yes, it sure will. The assembler's macros are even more powerful than I thought!"...Bill)

About Hardcore Magazine.....Bob Sander-Cederlof

I have received several calls by subscribers who wonder about the ad from Hardcore magazine. The ad prices a subscription at $20, but does not say clearly what $20 buys.

To my knowledge, HARDCORE has published two issues so far: the first one about a year ago, and the second about six months ago.

Inside the front cover of the first issue you will find the following message:

> "Attention Subscribers: Although presently only a quarterly magazine, HARDCORE Computing will go bimonthly and then monthly as soon as possible. Meanwhile, your one-year subscription is for the 4 quarterly issues plus 8 UPDATEs (printed on the other 8 months) and all ALERT Bulletins sent out whenever we feel information is too important to wait. The UPDATEs will be reprinted in part or in whole in the next magazine. The magazines, UPDATEs, and ALERT Bulletins comprise the subscription package."

I have talked with the publisher, Chuck Haight, several times on the phone. I believe he intends to fulfill every subscription, but he is having trouble getting the magazine out on a regular schedule. I asked him how often the magazine is published, and he answered "Very infrequently". He did re-assure me that a subscription buys four issues.

Note that Softkey Publishing is another company with the same people. Two callers indicated they are quite satisfied with the software they bought from Softkey.

No More Paddle Interaction.....Mike Laumer

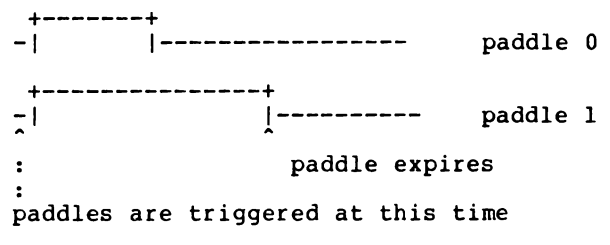
While working on the FLASH! Integer BASIC Compiler I ran into a nasty little problem because the compiled code ran too fast! That's right, too fast. The old problem with reading the game paddles too soon after one another rose to byte (punny huh!) me once again.

Basically the game paddle problem is that they are read with a variable time delay loop. Because one paddle may read significantly faster than another, and the paddles have only one trigger to fire all four of the paddles, you might process the data fast enough to be ready to read the next paddle before it has finished its previous time delay. This problem is real and occurs in many of the game programs to be found on the Apple. Even Raster Blaster has the problem in its jittery ball release thrust adjuster.

In the example below paddle 0 and 1 are triggered by the $C070 paddle I/O trigger address. But because paddle 0 has a smaller value, it finishes before paddle 1. If you read one paddle after another with little other processing then one paddle seems to affect the value of the other one. Many programmers have shown this problem to their dealer thinking that they have found a new bug in the Apple but the only problem (if one exists) is the lack of independent paddle triggers for each of the four paddles.

The problem appears if you use the following BASIC program and play with the paddle adjustments. Turn paddle 1 to the middle of its scale and paddle 0 to the low end of its scale and you will see changing paddle 0 affects the value read for paddle 1. You will find that paddle 1 will vary by 20-40 counts without even touching it.

```
10 PRINT PDL(0),PDL(1)   GOTO 10
```



So what can be done about the problem? What I did is design a routine that reads the paddle without triggering it and waits for the paddle to shut off. This is easily done by calling the monitor paddle read routine at $FB21, skipping the trigger instruction at $FB1E. This takes care of much of the problem, but I still found it necessary to add a tiny delay loop before triggering the paddle. The extra delay is probably due to the remaining charge in the internal capacitor in the timer chip.

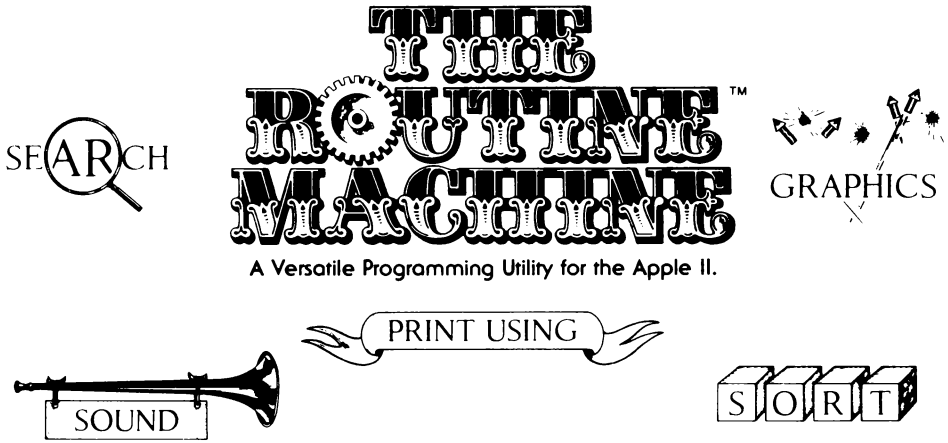
The assembly language routine which follows is basically what I added to the FLASH! compiler runtime package to take care of its being too fast for its own good! This explains 14 of the 36,000 bytes of object code in the FLASH! Compiler system. There is also a DEMO program which reads both paddles and displays the values in hexadecimal so you can test the routine.

```
                    1000 *--------------------------------
                    1010 * READ PADDLES
                    1020 * PADDLE NUMBER IN A REGISTER
                    1030 * USES A,X,Y REGISTERS
                    1040 * RETURNS PADDLE VALUE IN Y REGISTER
                    1050 *--------------------------------
                    1060 * THIS PADDLE READ ROUTINE
                    1070 * WILL PREVENT ALMOST ALL PADDLE
                    1080 * INTERACTION PROBLEMS DUE TO
                    1090 * ONLY 1 PADDLE TRIGGER FOR
                    1100 * ALL PADDLES.
                    1110 *--------------------------------
FB1E-               1120 MON.PREAD .EQ $FB1E
                    1130 *--------------------------------
0800- 29 03         1140 READP  AND #3            PDL 0 - 3
0802- AA            1150        TAX
0803- 20 21 FB      1160        JSR MON.PREAD+3   MAKE SURE PADDLE IS READY
0806- A0 00         1170        LDY #0
0808- 88            1180 .1     DEY               KLUDGE DELAY FOR
0809- D0 FD         1190        BNE .1              CIRCUIT READY
080B- 4C 1E FB      1200        JMP MON.PREAD     TRIGGER AND READ
                    1210 * PADDLE RESULT IN Y REGISTER
                    1220 *--------------------------------
080E- A9 00         1230 DEMO   LDA #0          READ PADDLE 0
0810- 85 24         1240        STA $24         HTAB COLUMN 1
0812- 20 00 08      1250        JSR READP
0815- 98            1255        TYA             VALUE TO A
0816- 20 DA FD      1260        JSR $FDDA       PRINT VALUE IN HEX
0819- E6 24         1270        INC $24         LEAVE SPACE ON SCREEN
081B- A9 01         1280        LDA #1          READ PADDLE 1
081D- 20 00 08      1290        JSR READP
0820- 98            1295        TYA             VALUE TO A
0821- 20 DA FD      1300        JSR $FDDA       PRINT VALUE IN HEX
0824- 4C 0E 08      1310        JMP DEMO        AGAIN AND AGAIN...
```

An Apple Bibliography

Bob Broedel has been keeping track of all the books, magazines, etc. that are of interest to Apple owners. The last time I saw the list (May 1982), it was ten pages, two columns. Each entry includes all the bibliographic data Bob knows, so that you can find the items you want.

This is the most complete list I have ever seen. If you want a copy, he will send you one for $2. Write to Bob Broedel, P. O. Box 20049, Tallahassee, FL 32304.



Now, from the programming experts at S.D.S., an easy-to-use way of putting the POWER and SPEED of machine language routines in YOUR OWN APPLESOFT PROGRAMS!

ROUTINE MACHINE does all the work for you — no knowledge of machine language programming, whatsoever, is required. Simply choose the routine you need from an ever-growing library, and Routine Machine will effortlessly put them in your program, and all done transparently! No need to be aware of or bother with BLOAD's, HIMEM:, etc.

Best of all, with just this starter package, you'll have the routines to put High Resolution **graphics** and **sound** in your programs immediately! Also included is a versatile **print using** module to banish the "decimal point demons" forever! To round out the package, we've also included powerful **search** and **sort** routines (for single dimension arrays; Search: 1000 elements in 1 second. Sort: 1000 elements in 90 seconds), and a number of other often-needed routines as well (30 routines in all).

OUR GUARANTEE

IF YOU DON'T SAVE MORE THAN THE PURCHASE PRICE OF 'ROUTINE MACHINE' IN YOUR OWN PROGRAMMING TIME IN THE FIRST 30 DAYS YOU OWN IT, SIMPLY RETURN IT FOR A COMPLETE REFUND, NO QUESTIONS ASKED!

Additional library disks titled **"Ampersoft Program Library"** are already available.

Some of the other routines in The Routine Machine (plus others not listed) are:

SWAP: Swaps two string or numeric values.

TEXT OUTPUT: Prints with no "word break" on screen.

STRING OUTPUT: Input any string, regardless of commas, etc.

ERR: Stack fix for Applesoft ONERR handling.

GOTO, GOSUB: Allows computed statements. Example: **GOTO X * 5** or **GOSUB X * 5.**

BLOAD: Load any binary file 5 times faster than normal. Hi-Res pictures load in under 2 seconds.

RESET HANDLER: Treats RESET with ONERR; or will RUN or reboot disk.

HI-RES ASCII: Character set for mixing text Hi-Res graphics.

TURTLE GRAPHICS: Versatile Hi-Res graphics routines for easy drawing of Hi-Res figures.

southwestern data systems™

P.O. BOX 582 • SANTEE, CALIFORNIA 92071 • TELEPHONE: 714/562-3670

Some Fast Screen Tricks.....Bob Sander-Cederlof

Sometimes the standard Apple Monitor screen functions are too slow. No reflection on Steve Wozniak, because he wrote them to be general and compact rather than quick.

I am thinking particular of the screen clear (HOME to Applesoft users) and the screen scroll subroutines. They were both written to operate on a text window, not necessarily the whole screen. But most of the time you do want to clear or scroll the whole screen.

The primary text screen memory is mapped into the addresses from $400 through $7FF, but not in an obvious or straightforward way. This table shows the actual memory addresses for each screen line:

Line	Addresses	Line	Addresses	Line	Addresses
0	$400-$427	8	$428-$44F	16	$450-$477
1	$480-$4A7	9	$4A8-$4CF	17	$4D0-$4F7
2	$500-$527	10	$528-$54F	18	$550-$577
3	$580-$5A7	11	$5A8-$5CF	19	$5D0-$5F7
4	$600-$627	12	$628-$64F	20	$650-$677
5	$680-$6A7	13	$6A8-$6CF	21	$6D0-$6F7
6	$700-$727	14	$728-$74F	22	$750-$777
7	$780-$6A7	15	$7A8-$7CF	23	$7D0-$7F7

Note that 120 consecutive bytes are used for three text lines spaced at an 8-line interval. Then 8 bytes are not used. Then the next 120, and so on. Those 8 sets of 8 bytes that are not used by the screen mapping are used by peripheral cards and DOS for temporary storage. In the standard Apple Monitor subroutines, a subroutine named BASCALC at $FBC1 calculates the starting address for a specified line. Then the various screen functions use that address, which is kept up-to-date in BASL,BASH ($28,29).

In the listing that follows, I have included fast subroutines to clear the entire text screen (CLEAR); to set the entire text screen to whatever character is in the A-register (SET); to clear the entire Lo-Res Graphics screen (GCLEAR); and to scroll the entire text screen up one line. For demonstration purposes, I also wrote routines to set the entire screen to each value from $00 through $FF; to alternate the screen between solid black and solid white until a key is pressed; to scroll end-around, placing the old top line on the bottom of the screen while moving the rest of the lines up; and to continuously scroll end-around until a key is pressed.

For comparison, I counted that the Wozniak's screen clear takes 15537 microseconds; mine takes only 5410 microseconds. The fastest possible would be one LDA #$A0 followed by 960 "STA $xxx" and an RTS; that would take 3848 microseconds. (All these times round off the Apple's cycle time to one microsecond; actually it is a little faster.)

```
                    1000 * S.SCREEN TRICKS
                    1010 *--------------------------------
                    1020 *    FAST SCREEN CLEAR SUBROUTINE
                    1030 *--------------------------------
0800- A9 00         1040 GCLEAR LDA #0
0802- 2C            1050        .HS 2C          SKIP OVER NEXT TWO BYTES
0803- A9 A0         1060 CLEAR  LDA #$A0
0805- A0 77         1070 SET    LDY #119
0807- 99 00 04      1080 .1     STA $400,Y      LINES:  0  8 16
080A- 99 00 05      1090        STA $500,Y              2 10 18
080D- 99 00 06      1100        STA $600,Y              4 12 20
0810- 99 00 07      1110        STA $700,Y              6 14 22
0813- 99 80 04      1120        STA $480,Y              1  9 17
0816- 99 80 05      1130        STA $580,Y              3 11 19
0819- 99 80 06      1140        STA $680,Y              5 13 21
081C- 99 80 07      1150        STA $780,Y              7 15 23
081F- 88            1160        DEY
0820- 10 E5         1170        BPL .1
0822- 60            1180        RTS
                    1190 *--------------------------------
                    1200 *      SET SCREEN TO ALL VALUES
                    1210 *--------------------------------
0823- A2 00         1220 SETALL LDX #0
0825- 8A            1230 .1     TXA
0826- 20 05 08      1240        JSR SET
0829- E8            1250        INX
082A- D0 F9         1260        BNE .1
082C- 60            1270        RTS
                    1280 *--------------------------------
                    1290 *   ALTERNATE SCREEN UNTIL KEY PRESSED
                    1300 *--------------------------------
082D- A9 20         1310 ALTER  LDA #$20        INVERSE BLANK
082F- 20 05 08      1320        JSR SET
0832- 20 03 08      1330        JSR CLEAR
0835- AD 00 C0      1340        LDA $C000
0838- 10 F3         1350        BPL ALTER
083A- 8D 10 C0      1360        STA $C010
083D- 60            1370        RTS
                    1380 *--------------------------------
                    1390 *   FAST SCROLL UP SUBROUTINE
                    1400 *--------------------------------
083E- A0 77         1410 SCROLL LDY #119
0840- B9 00 04      1420 .1     LDA $400,Y      SAVE LINES: 0 8 16
0843- 48            1430        PHA
0844- B9 80 04      1440        LDA $480,Y      MOVE 1>0, 9>8, 17>16
0847- 99 00 04      1450        STA $400,Y
084A- B9 00 05      1460        LDA $500,Y      MOVE 2>1, 10>9, 18>17
084D- 99 80 04      1470        STA $480,Y
0850- B9 80 05      1480        LDA $580,Y      MOVE 3>2, 11>10, 19>18
0853- 99 00 05      1490        STA $500,Y
0856- B9 00 06      1500        LDA $600,Y      MOVE 4>3, 12>11, 20>19
0859- 99 80 05      1510        STA $580,Y
085C- B9 80 06      1520        LDA $680,Y          ET CETERA
085F- 99 00 06      1530        STA $600,Y
0862- B9 00 07      1540        LDA $700,Y
0865- 99 80 06      1550        STA $680,Y
0868- B9 80 07      1560        LDA $780,Y
086B- 99 00 07      1570        STA $700,Y
086E- 68            1580        PLA             MOVE 8>7, 16>15
086F- C0 28         1590        CPY #40
0871- 90 03         1600        BCC .2          DISCARD OLD LINE 0
0873- 99 58 07      1610        STA $780-40,Y
0876- 88            1620 .2     DEY
0877- 10 C7         1630        BPL .1
0879- 60            1640        RTS
                    1650 *--------------------------------
                    1660 *   SCROLL AROUND, MOVING TOP LINE TO BOTTOM
                    1670 *--------------------------------
087A- A0 27         1680 SCR    LDY #39         SAVE TOP LINE ON STACK
087C- B9 00 04      1690 .1     LDA $400,Y
087F- 48            1700        PHA
0880- 88            1710        DEY
0881- 10 F9         1720        BPL .1
0883- 20 3E 08      1730        JSR SCROLL      SCROLL SCREEN UP ONE LINE
0886- A0 00         1740        LDY #0          STORE OLD TOP LINE
0888- 68            1750 .2     PLA                ON BOTTOM OF SCREEN
0889- 99 D0 07      1760        STA $7D0,Y
088C- C8            1770        INY
088D- C0 28         1780        CPY #40
088F- 90 F7         1790        BCC .2
0891- 60            1800        RTS
                    1810 *--------------------------------
                    1820 *   ROTATE SCREEN UNTIL KEY PRESSED
                    1830 *--------------------------------
0892- 20 7A 08      1840 S      JSR SCR         SCROLL AROUND ONCE
0895- AD 00 C0      1850        LDA $C000       ANY KEY PRESSED?
0898- 10 F8         1860        BPL S           NO, SCROLL AGAIN
089A- 8D 10 C0      1870        STA $C010       YES, CLEAR STROBE
089D- 60            1880        RTS             ...AND RETURN
```

Right arrow for the VIDEX patches.....Mike Laumer

The VIDEX 80 column board patches for the S-C Macro Assembler in last months Apple Assembly Line was a welcome article for me. You see I bought a VIDEX board last November but have no software to run it. I've been planning to write a program development editor similar to the one I used at Texas Instruments, but so far I haven't had the time between the FLASH! compiler, MIKE'S MAGIC MATRIX and the American Heart association CPR Training system.

The patches were very usable, but a major problem still existed to prevent my use on a regular basis. The right arrow key would not copy characters from the VIDEX screen. Try to copy a file name from your catalog with that limitation!

I knew it could be done, because the VIDEX software in ROM has to do that function. Don Taylor mentioned last month that he didnt know the right routine to call and his ROM differed from the listing in the VIDEX manual. My listing was a little off also from my ROM, but I didn't care becase I wasn't going to call the ROM routines.

I used the VIDEX manual's listings to locate the section that performed the copy-character-from-screen function and used similar code in the RDKEY routine of last month's VIDEX patches for the Macro assembler. The 'BNE' to '.3' was changed to go to 'CTRLU' and the copy function coded to process the right arrow key for the VIDEX 80 column board.

I needed two temporary variables to save the X- and Y-registers, so I used the first two bytes of the normal Apple text screen at $400 and $401. Another temporary variable is at $402. Since the normal Apple text display is not operative while the VIDEX is enabled you can use it for temporary variable space without it affecting the screen display. If you try a trick like this some time, you must be careful because some of the monitor routines like HOME and SCROLL can easily zap your storage when you least expect it.

With this new capability of the right arrow key functioning as expected, I am able to use the VIDEX patches often in my software development work. But there are a few problems left yet to solve that I didn't get to look into before writing this article. They are:

1. A RETURN key should clear to the end of line on line input, but not EDIT input.

2. The control character display features are not handled very well by the VIDEX patches.

3. The patches blow up on Reset. (I think.)

4. The patches blow up on INT or FP commands.

5. The patches don't work very well when you use MNTR command.

6. All calls to $FC9C (the Monitor clear to end of line routine) should send $9D to the VIDEX board.

7. Right arrow, left arrow, and any printing key cause the entire EDIT line to be redisplayed. The flicker is somewhat annoying.

The listing that follows should replace lines 4020 through 4420 of the listing on pages 21 and 22 of the August 1982 issue.

The source code on the AAL Quarterly Disk #8 will have these lines already merged with Don Taylor's patches.

Special Note: S-C Macro Cross Assembler
Motorola 6800/6801/6802 Version

The 6801 microprocessor is an enhanced version of the 6800 cpu. It has 11 new opcodes, plus an additional addressing mode for the JSR instruction. Be sure not to use any of these new opcodes if you are assembling code which must execute in a 6800 system!

```
JSR direct  (9D xx)
ABX     Add B to X
ADDD    Add M,M+1 to D
ASLD    Arithmetic shift left D
BRN     Branch Never
LDD     Load D from M,M+1
LSRD    Logical shift right D
MUL     Multiply A * B into D
PSHX    Push X
PULX    Pull X
STD     Store D at M,M+1
SUBD    Subtract M,M+1 from D
```

If you attempt to use "JSR addr" where "addr" is in page zero, the new direct addressing mode will be used. If you are programming for a 6800 cpu, that is not acceptable. To override the assembler's choice, use ".DA #$BD,addr". You can use a macro "JSR" if you have a lot of them.

```
4020 *--------------------------------
4025 V.BASEL .EQ $478+SLOTNUM
4030 V.BASEH .EQ $4F8+SLOTNUM
4035 V.CHORZ .EQ $578+SLOTNUM
4040 V.XSAV1 .EQ $402
4045 V.OLDCHAR .EQ $678
4050 *
4055 V.DEVO  .EQ SLOTNUM*16+$C080
4060 V.DISPO .EQ $CC00
4065 V.DISP1 .EQ $CD00
4070 *--------------------------------
4075 *
4080 RDKEY   LDA KEYBOARD
4085         BPL RDKEY
4090         STA KEYSTROBE
4095         ORA #$80
4100         CMP #$81      Shift lock?
4105         BNE .1
4110         .DO LCVERSION
4115         JSR UNPROTECT.LC.RAM
4120         .FIN
4125         LSR SCM.SHIFT.FLAG
4130         BPL .2        Return with errant key
4135 .1      CMP #$9A      Shift unlock?
4140         BNE CTRLU     No, return with key
4145         .DO LCVERSION
4150         JSR UNPROTECT.LC.RAM
4155         .FIN
4160         SEC
4165         ROR SCM.SHIFT.FLAG
4170 .2      LDA #$96      Return with errant key
4175         .DO LCVERSION
4180         BIT $C080     Reprotect LC RAM
4185         RTS
4190 *
4195 UNPROTECT.LC.RAM
4200         BIT $C083     Enable Bank 2
4205         BIT $C083
4210         .FIN
4215         RTS
4220 *
4225 CTRLU   CMP #$95      CTRL-U COPY KEY
4230         BNE .3
4235         STX $400
4240         STY $401
4245         LDA V.CHORZ
4250         JSR PSNCALC
4255         BCS .1
4260         LDA V.DISPO,X
4265         BCC .2
4270 .1      LDA V.DISP1,X
4275 .2      ORA #$80
4280         STA V.OLDCHAR
4285         LDX $400
4290         LDY $401
4295 .3      RTS
4300 *
4305 PSNCALC CLC
4310         ADC V.BASEL
4315         STA V.XSAV1
4320         LDA #0
4325         ADC V.BASEH
4330         LSR
4335         PHP
4340         AND #3
4345         ASL
4350         ASL
4355         TAY
4360         LDA V.DEVO,Y
4365         PLP
4370         LDX V.XSAV1
4375         RTS
4380 *--------------------------------
```

A Note on the Underline Cursor.....Bob Sander-Cederlof

Bill Linn's "Blinking Underline Cursor" program generated a lot of interest. However, Allan Blackburn from Fort Worth had a problem with it:

> "It works just fine, until you hit RESET or re-boot...then it must be BRUN again to get it back. You can't enter monitor and type 300G, or use CALL 768 from Applesoft. Why doesn't calling the routine reset KSWL and KSWH? It should, but I always end up with $9E81 there. Even though lines 1210-1250 store $09 in $38 and $03 in $39, it seems they never get there. Can you explain this? Please?

Sure, Allan. Line 1250 needs to be changed from RTS to JMP $3EA.

This is a common problem. I had it myself back when DOS first came out. For the first year or so we only had a tiny preliminary manual, and the subject wasn't covered. Now the DOS manual is so large we forget to read it or where to find the information. Look on pages 100-105 of the DOS manual and you will find a full explanation.

Briefly, here is what happens. Lines 1210-1250 DO store the address $309 into #38 and $39. But the next time you print a character, DOS gets control and stores its own input address right on top of yours. DOS's input address is $9E81.

The same thing happens in Applesoft programs if you use IN#1 (for example) instead of PRINT CHR$(4)"IN#1", and then print a character. Note 7b on page 105 tells about CALL 1002, which is $3EA.

Apple Assembly Line is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228. Phone (214) 324-2050. Subscription rate is $15 per year in the USA, sent Bulk Mail; add $3 for First Class postage in USA, Canada, and Mexico; add $13 postage for other countries. Back issues are available for $1.50 each (other countries add $1 per back issue for postage).